DATA MINING & DISCOVERY

SQL ASSIGNMENT

SUBMITTED BY:

NAME: Saranya Bala Subramaniam

STUDENT ID: 22033095

PYTHON SCRIPT THAT GENERATED THE DATABASE:

```
import csv
import random
from faker import Faker
import sqlite3
import os
fake = Faker()

# Function to generate random ordinal data
def generate_ordinal_data():
    ratings = ["Low", "Medium", "High"]
    return random.choice(ratings)

# Function to generate random nominal data
def generate_nominal_data():
    categories = ["Electronics", "Clothing", "Home Appliances"]
    return random.choice(categories)

# Function to generate random ratio data
def generate_ratio_data():
    return round(random.uniform(10.0, 500.0), 2)

# Function to generate random interval data
def generate_interval_data():
    return random.randint(1, 10)

# Set the current working directory to the script's directory
script_directory = os.path.dirname(os.path.realpath(__file__))
os.chdir(script_directory)
```

```
# Generate Products data
products_data = []
for product_id in range(1, 1001):
    category = generate_nominal_data()
    price = generate_ratio_data()
    stock_quantity = random.randint(20, 100)
    product_rating = generate_ordinal_data()
    customer_id = fake.uuid4()  # Adding customer_id
        product = {
        "product_id": product_id,
        "product_name": fake.word(),
        "category": category,
        "price": price,
        "stock_quantity": stock_quantity,
        "product_rating": product_rating,
        "customer_id": customer_id,
    }
    products_data.append(product)

# Generate Sales data
sales_data = []
for sale_id in range(1, 1001):
    customer_name = fake.name()
    sale_date = fake.date_this_year()
    customer_id = fake.uuid4()  # Adding customer_id
        sale = {
        "sale_id": sale_id,
        "customer_id": customer_id,
        "customer_name": customer_name,
        "sale_date": sale_date,
    }
```

```
    sales_data.append(sale)

# Generate Sales Details data (Child Table)
sales_details_data = []
for sale_id in range(1, 1001):
    product_id = random.randint(1, 1000)
    quantity_sold = random.randint(1, 10)
        sales_details = {
        "sale_id": sale_id,
        "product_id": product_id,
        "quantity_sold": quantity_sold,
    }
    sales_details_data.append(sales_details)

# Define CSV file names with full paths
products_csv_file = os.path.join(script_directory, "products_data.csv")
sales_csv_file = os.path.join(script_directory, "sales_data.csv")
sales_details_csv_file = os.path.join(script_directory, "sales_details_data.csv")

# Write data to CSV files
with open(products_csv_file, mode="w", newline="") as file:
    fieldnames = ["product_id", "product_name", "category", "price", "stock_quantity",
"product_rating", "customer_id"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(products_data)
with open(sales_csv_file, mode="w", newline="") as file:
    fieldnames = ["sale_id", "customer_id", "customer_name", "sale_date"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(sales_data)
```

```
with open(sales_details_csv_file, mode="w", newline="") as file:
    fieldnames = ["sale_id", "product_id", "quantity_sold"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(sales_details_data)

# Function to create SQLite database and import data from CSV files
def create_sales_database(products_csv, sales_csv, sales_details_csv, database_name):
    conn = sqlite3.connect(database_name)
    cursor = conn.cursor()

    # Create 'products' table if it does not exist
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND
name='products'")
    if not cursor.fetchone():
        with open(products_csv, 'r') as file:
            reader = csv.reader(file)
            header = next(reader)
            columns = ', '.join(header)
            cursor.execute(f'CREATE TABLE products ({columns})')
            cursor.executemany(f'INSERT INTO products VALUES ({", ".join(["?"] * len(header))})',
reader)

    # Create 'sales' table if it does not exist
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND
name='sales'")
    if not cursor.fetchone():
        with open(sales_csv, 'r') as file:
            reader = csv.reader(file)
            header = next(reader)
            columns = ', '.join(header)
            cursor.execute(f'CREATE TABLE sales ({columns})')
```

```
        cursor.executemany(f'INSERT INTO sales VALUES ({", ".join(["?"] * len(header))})', reader)


    # Create 'sales_details' table if it does not exist
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='sales_details'")
    if not cursor.fetchone():
        with open(sales_details_csv, 'r') as file:
            reader = csv.reader(file)
            header = next(reader)
            columns = ', '.join(header)
            cursor.execute(f'CREATE TABLE sales_details ({columns})')
            cursor.executemany(f'INSERT INTO sales_details VALUES ({", ".join(["?"] * len(header))})', reader)


    # Commit changes and close connection
    conn.commit()
    conn.close()


# Create the 'sales_database.db' and import data
create_sales_database(products_csv_file, sales_csv_file, sales_details_csv_file, 'sales_database.db')
```

1.Data Generation:

- *Using the Faker package and random functions, the given Python script creates fake data for a sales database.*
- *The generated data includes information about products, sales, and sales details.*
- *Product categories, prices, and stock quantities are representative of a fictional sales scenario. Customer names, product names, and sale dates are randomly generated.*
- *The synthetic data generated in this script can be used for testing and development purposes, simulating a sales database with related tables for products and sales details.*

Below is an explanation of the data generation process:

1. Ordinal Data (Product Ratings):

```
# Function to generate random ordinal data
def generate_ordinal_data():
    ratings = ["Low", "Medium", "High"]
    return random.choice(ratings)
```

2. Nominal Data (Product Categories):

```
# Function to generate random nominal data
def generate_nominal_data():
    categories = ["Electronics", "Clothing", "Home Appliances"]
    return random.choice(categories)
```

3. Ratio Data (Product Price):

```
# Function to generate random ratio data
def generate_ratio_data():
    return round(random.uniform(10.0, 500.0), 2)
```

4. Interval Data (Stock Quantity):

```
# Function to generate random interval data
def generate_interval_data():
    return random.randint(1, 10)
```

5. UUID Generation (Customer ID):

- Customer IDs are generated using the fake.uuid4() function from the Faker library.

6. Date Generation (Sale Date):

- Sale dates are generated using the fake.date_this_year() function from the Faker library.

7. Random Product Names and Customer Names:

- *Product names and customer names are generated using the fake.word() and fake.name() functions, respectively.*

CSV File Creation:

- *Three CSV files are created: "products_data.csv," "sales_data.csv," and "sales_details_data.csv."*
- *Data for products, sales, and sales details are written to these files using the `csv.DictWriter` module.*

SQLite Database Creation:

- *A SQLite database named "sales_database.db" is created.*
- *The script defines a function, `create_sales_database`, to create tables based on the CSV file structures and import data into them.*
- *The function is called for each CSV file to create tables for products, sales, and sales details.*

2.THE SCHEMA OF THE DATABASE:

The schema establishes relationships between the tables using foreign keys and ensures data integrity through primary keys.

Table 1- products:

```
CREATE TABLE "products" (
        "product_id"    INTEGER NOT NULL,
        "product_name"        TEXT,
        "category"      TEXT,
        "price" REAL,
        "stock_quantity"  INTEGER,
        "product_rating"        TEXT,
        "customer_id" TEXT NOT NULL,
        PRIMARY KEY("product_id"),
        FOREIGN KEY("customer_id") REFERENCES "sales"("customer_id")
);
```

'product_id' is the primary key, and 'customer_id' is a foreign key referencing the 'customer_id' in the 'sales' table.

- *product_id: An integer representing the unique identifier for each product. It is marked as NOT NULL, indicating it must have a value, and is set as the primary key for this table.*
- *product_name: A text field storing the name of the product.*
- *category: A text field indicating the category of the product.*
- *price: A real number representing the price of the product.*
- *stock_quantity: An integer representing the quantity of the product in stock.*
- *product_rating: A text field representing the rating of the product.*
- *customer_id: A text field representing the customer ID to which this product is associated. It is marked as NOT NULL, indicating it must have a value, and it is a foreign key referencing the customer_id in the sales table.*

Table 2- sales:

```
CREATE TABLE "sales" (
        "sale_id"       INTEGER NOT NULL,
        "customer_id" TEXT NOT NULL,
        "customer_name"      TEXT,
        "sale_date"     DATE,
        PRIMARY KEY("sale_id","customer_id"),
        FOREIGN KEY("customer_id") REFERENCES "products"("customer_id")
);
```

the primary key is a composite key consisting of 'sale_id' and 'customer_id'. This means that the combination of sale_id and customer_id must be unique. The 'customer_id' is a foreign key referencing the 'customer_id' in the 'products' table.

- *sale_id: An integer representing the unique identifier for each sale. It is marked as NOT NULL, indicating it must have a value, and it is part of the primary key for this table.*
- *customer_id: A text field representing the customer ID associated with this sale. It is marked as NOT NULL, indicating it must have a value, and it is a foreign key referencing the customer_id in the products table.*
- *customer_name: A text field storing the name of the customer.*
- *sale_date: A date field indicating the date of the sale.*

Table 3- sales details:

```
CREATE TABLE "sales_details" (
        "sale_id"       INTEGER NOT NULL,
        "product_id"    INTEGER NOT NULL,
        "quantity_sold"         INTEGER,
        PRIMARY KEY("product_id","sale_id"),
        FOREIGN KEY("product_id") REFERENCES "products"("product_id"),
        FOREIGN KEY("sale_id") REFERENCES "sales"("sale_id")
);
```

the primary key is a composite key consisting of `product_id` and `sale_id`. The `product_id` is a foreign key referencing the `product_id` in the `products` table, and the `sale_id` is a foreign key referencing the `sale_id` in the `sales` table.

- *sale_id: An integer representing the unique identifier for each sale detail. It is marked as NOT NULL, indicating it must have a value, and it is part of the primary key for this table. It is also a foreign key referencing the sale_id in the sales table.*
- *product_id: An integer representing the unique identifier for each product detail. It is marked as NOT NULL, indicating it must have a value, and it is part of the primary key for this table. It is also a foreign key referencing the product_id in the products table.*
- *quantity_sold: An integer representing the quantity of the product sold in the specific sale.*

3. JUSTIFICATION FOR ANY SEPARATE TABLE AND ETHICAL DISCUSSION:

By minimising redundancy and adhering to normalisation standards, this separation protects data integrity through relationships.

***Products Table:** Contains information about individual products such as product_id, product_name, category, price, stock_quantity, product_rating, and customer_id.*

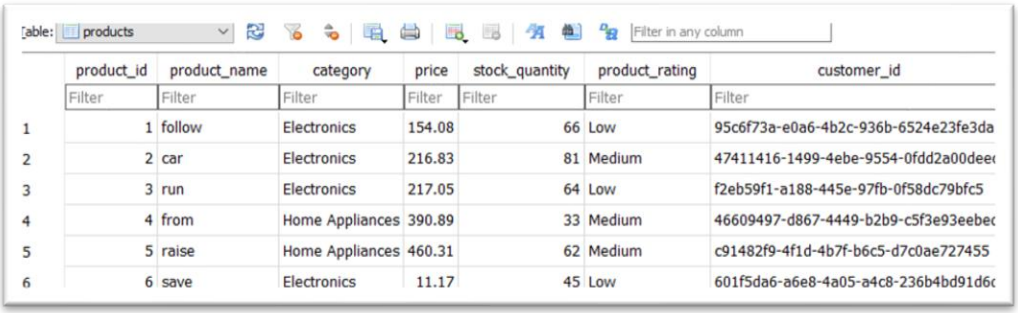
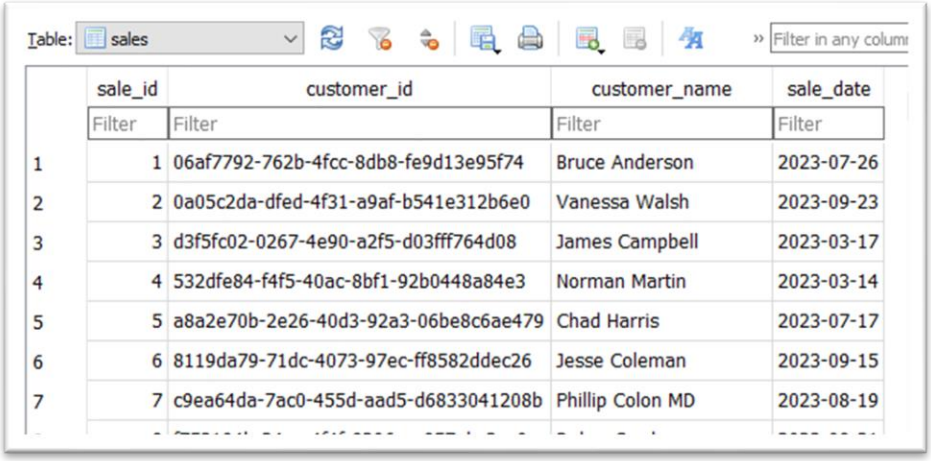


Table: products

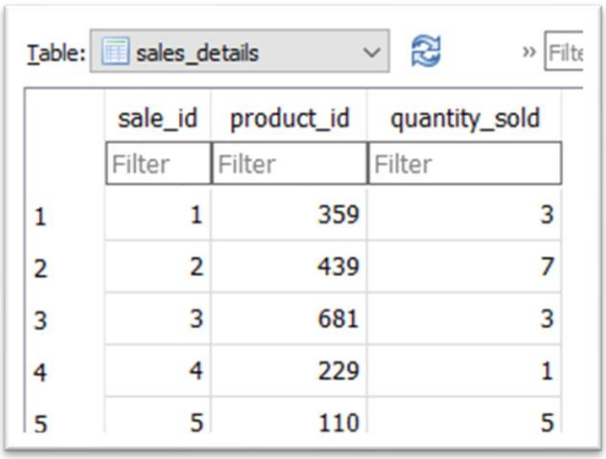
Filter in any column

	product_id	product_name	category	price	stock_quantity	product_rating	customer_id
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	follow	Electronics	154.08	66	Low	95c6f73a-e0a6-4b2c-936b-6524e23fe3da
2	2	car	Electronics	216.83	81	Medium	47411416-1499-4ebe-9554-0fdd2a00deec
3	3	run	Electronics	217.05	64	Low	f2eb59f1-a188-445e-97fb-0f58dc79bfc5
4	4	from	Home Appliances	390.89	33	Medium	46609497-d867-4449-b2b9-c5f3e93eebec
5	5	raise	Home Appliances	460.31	62	Medium	c91482f9-4f1d-4b7f-b6c5-d7c0ae727455
6	6	save	Electronics	11.17	45	Low	601f5da6-a6e8-4a05-a4c8-236b4bd91d6c

***Sales Table:** Stores information about sales transactions, including sale_id, customer_id, customer_name, and sale_date.*



Sales Details Table: *Captures details about each sale, linking sale_id to product_id and recording quantity_sold.*



By minimising redundancy and adhering to normalisation standards, this separation protects data integrity through relationships.

Flexibility and Maintenance:

Products*: It is simple to edit product details without compromising sales data.*

Sales*: Individual product details are unaffected by changes to client information or sale dates.*

Sales details: *Allows for changes to amounts sold without affecting essential product or sales data.*

The system is more versatile and easier to maintain because to its modular architecture.

Privacy and Security:

- ***Products****: Ethical considerations include safeguarding product data.*
- ***Sales****: Protects customer-related information.*
- ***Sales Details****: Ensures transaction details are secure.*

Robust security measures are vital to prevent unauthorized access and protect sensitive data.

Consent and Transparency:

- ***Products****: Users should be aware of data collection related to products.*
- ***Sales****: Transparency in sales transactions, with clear communication about data usage.*
- ***Sales Details****: Users need to understand how transaction details are recorded.*

Transparency builds trust and helps users make informed decisions.

Data Accuracy and Fairness:

- ***Products****: Ensuring accurate representation of products.*
- ***Sales****: Accuracy in recording sales transactions.*
- ***Sales Details****: Precision in capturing quantities sold.*

Ethical responsibility involves maintaining accurate and fair data representation.

4. Example queries of your database including joins and selections, demonstrating different data types.

Query 1 - *Selecting Products with Price Greater Than $200:*

```
1  SELECT * FROM products WHERE price > 200.0;
2
```

	product_id	product_name	category	price	stock_quantity	product_rating	customer_id
1	2	car	Electronics	216.83	81	Medium	47411416-1499-4ebe-9554-0fdd2a00deec
2	3	run	Electronics	217.05	64	Low	f2eb59f1-a188-445e-97fb-0f58dc79bfc5
3	4	from	Home Appliances	390.89	33	Medium	46609497-d867-4449-b2b9-c5f3e93eebed
4	5	raise	Home Appliances	460.31	62	Medium	c91482f9-4f1d-4b7f-b6c5-d7c0ae727455
5	7	address	Clothing	222.34	28	Medium	1f020a89-e187-4fc3-a958-6954e9dbff8f

Query 2 - *Joining Sales and Sales Details to Get Customer Names and Quantity Sold:*

```
1  SELECT s.sale_id, s.customer_name, sd.quantity_sold
2  FROM sales s
3  JOIN sales_details sd ON s.sale_id = sd.sale_id;
4
```

	sale_id	customer_name	quantity_sold
1	1	Bruce Anderson	3
2	2	Vanessa Walsh	7
3	3	James Campbell	3
4	4	Norman Martin	1
5	5	Chad Harris	5
6	6	Jesse Coleman	4

Query 3 - *Aggregating Sales Details to Get Total Quantity Sold for Each Product:*

```
SELECT product_id, SUM(quantity_sold) AS total_quantity_sold
FROM sales_details
GROUP BY product_id;
```

	product_id	total_quantity_sold
1	2	8
2	3	4
3	5	3
4	7	8
5	9	1
6	10	10